EXHIBIT N

Contents

HTTP Cookies: Standards, Privacy, and Politics

David M. Kristol
Bell Labs, Lucent Technologies

How did we get from a world where cookies were something you ate and where "non-techies" were unaware of "Netscape cookies" to a world where cookies are a hot-button privacy issue for many computer users? This paper will describe how HTTP "cookies" work, and how Netscape's original specification evolved into an IETF Proposed Standard. I will also offer a personal perspective on how what began as a straightforward technical specification turned into a political flashpoint when it tried to address non-technical issues such as privacy.

Categories and Subject Descriptors: C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Applications*; K.2 [**History of Computing**]: Systems

General Terms: Standardization, Security, Design

Additional Key Words and Phrases: Cookies, state management, HTTP, privacy, World Wide Web

1. INTRODUCTION

The topic of HTTP "cookies" has become at least slightly familiar to many Internet users. Articles in the popular press now regularly mention cookies in conjunction with privacy concerns. However, cookies were in use for over two years before they first achieved notoriety, and some of that notoriety emerged around the same time as the appearance of the first formal standard for cookies, which had previously been informally described on Netscape's Communications Corporation's web site.

The cookie standardization process began in April, 1995, with a discussion on [www-talk]. In December of that year, the IETF undertook to write a cookie standard. After a series of Internet Drafts got published in connection with extensive public discussion on [http-wg] (and after noticeable delays due to IETF process), RFC 2109 [Kristol and Montulli 1997], *HTTP State Management Mechanism*, was published in February, 1997 as an IETF Proposed Standard. Technical and political concerns immediately emerged that led to further discussions and revisions (and delays) and that finally culminated, in October, 2000, in the publishing of RFC 2965 [Kristol and Montulli 2000].

In this paper I describe what cookies are, how they work, and how applications use them. I also relate the history of how cookies came to be standardized, and how the protracted process of standardization interacted with other forces in the explosive early evolution of the World Wide Web. We participants in the standardization process unexpectedly found ourselves at the intersection of technology and public policy when the proposed standard raised concerns about privacy. As co-editor of

Address: 600 Mountain Ave., Murray Hill, NJ 07974

the specification, I'll reflect on what happened.

2. WHAT ARE COOKIES? WHY ARE THEY USEFUL?

Any discussion of cookies must begin by answering two questions:[1] What are they? Why are they needed? The answers require a modest understanding of how the World Wide Web (WWW or "Web") works, which the next section provides.

2.1 An Introduction to Hypertext Transfer Protocol (HTTP)

The Hypertext Transfer Protocol (HTTP [Fielding et al. 1999]) provides the foundation for the Web, and cookies are an addition to HTTP. When a user clicks on a (hypertext) link in a web browser, the browser (sometimes referred to as "client" or "user agent") typically connects to the web server identified by the Uniform Resource Locator (URL) embedded in the link and sends it a request message, to which the server sends a response message. Then, after receiving the response, the browser disconnects from the server. Because the client makes a new connection for each request, the server treats each request as though it were the first one it had received from that client. We therefore consider the request to be "stateless:" each request is treated completely independently of any previous one.[2]

Statelessness makes it easier to build web browsers and servers, but it makes some web applications harder to write. For example, it would have been much harder to create the now-ubiquitous web shopping applications if they could not keep track of what's in your shopping basket.

HTTP requests (responses) comprise three parts:

(1) a request (response) line;
(2) request (response) headers, which provide meta-information; and
(3) the request (response) entity itself.

The header meta-information provides both control information for HTTP and information about the entity being transferred. Information about cookies gets conveyed in such headers.

Here is an example request. The first line is the request line. The remaining lines are request headers. There is no entity for a GET request.

```
GET / HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg,
  image/pjpeg, application/vnd.ms-powerpoint,
  application/vnd.ms-excel, application/msword, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows 98;
   Win 9x 4.90)
```

[1] Inevitably there's a third question: Where did the term "cookie" come from, anyway? *Magic cookie*, or just *cookie*, is a computer jargon term with a long and honorable history; it refers to an opaque identifier that gets passed back and forth between different pieces of software [Raymond 1996].

[2] Newer clients and servers are able to maintain a connection for more than one request-response cycle, but the essential stateless behavior remains.

```
Host: aleatory.research.bell-labs.com:80
Connection: Keep-Alive
```
---blank line---

The corresponding response might look like this.

```
HTTP/1.1 200 OK
Date: Thu, 25 Jan 2001 16:40:54 GMT
Server: Apache/1.3.12 (Unix)
Last-Modified: Fri, 05 Jan 2001 23:38:49 GMT
ETag: "121be7-15d-3a565b09"
Accept-Ranges: bytes
Content-Length: 1706
Content-Type: text/html
```
---blank line---
---HTML entity here---

Note that the request and response information is readable text, although the entity transferred need not be. "Web pages" are often encoded in Hypertext Markup Language (HTML), as in this example (although the actual HTML content has been omitted).

2.2 How Do Cookies Work?

Web-based applications often use cookies to maintain state in the otherwise stateless HTTP protocol. As part of its response, a server may send arbitrary information, the "cookie," in a `Set-Cookie` response header. This arbitrary information could be anything: a user identifier, a database key, whatever the server needs so it can continue where it left off. Under normal circumstances (and simplifying greatly), a cooperating client returns the cookie information verbatim in a `Cookie` header, one of its request headers, each time it makes a new request to the same server. The server may choose to include a new cookie with its responses, which would supersede the old one. Thus there is an implied "contract" between a server and client: the server relies on the client to save the server's state and to return it on the next visit.

To correct a frequent misstatement in early press stories, cookies do not arise from some insidious invasion of your computer or hard drive by an external intruder. Rather, your browser stores only those cookies it receives from a server it has visited. (However, it will be clear later that your browser may visit servers on your behalf without your knowing it and store cookies from them on your computer.)

A "cookie," then, is the piece of information that the server and client pass back and forth. The amount of information is usually small, and its content is at the discretion of the server. In general simply examining a cookie's value will not reveal what the cookie is for or what the value represents.

Restricting the client to return a cookie just to the server from which it was received is very limiting. Organizations often have multiple servers, and those servers need to have access to the same state information so that, in the aggregate, they can provide a service. Therefore, when it sends a cookie to a client, a server may specify, in a constrained way, the set of other servers to which a client may also send the cookie in subsequent requests.

The server can tell the browser to return cookies only in requests to specific parts of its site, as identified by the URLs. Provided applications use different parts of the "URL space" (for example, `http://shop.com/application1` and `http://shop.com/application2`) servers (or sets of servers) may thus host multiple cookie-based applications.[3]

2.3 Proxies

Users can configure their browsers to use an HTTP *proxy* for a variety of reasons, such as to improve performance, or because their Internet Service Provider (ISP) or company requires them to do so. An HTTP proxy is an intermediary that accepts a request from a client and forwards it to a server, then receives the resulting response and forwards it back to the client. A typical proxy accepts connections from multiple clients and can connect to any server. A pure HTTP proxy does not save the results of the requests and responses it forwards and poses no problems for applications that use cookies.

A *caching proxy*, however, may store responses. The purpose of a caching proxy is to reduce network traffic and response latency: the caching proxy may be able to return the same response to client 2 that it previously returned to client 1 for the same request, without the need to forward the second request to the origin server. However, in some cases returning the same response to both clients is the wrong thing to do, such as when the response for client 1 contains personalized content, or when the response is time-dependent. In such a case, the origin server must send response headers that direct any proxies between the server and the client (there may be zero or more proxies) not to cache the content, or to revalidate the response with the origin server before returning the response to the client.

Cookies and caching proxies can interact in undesirable ways, and cookie-based web applications must take that possibility into account. For example, a shopping site may allow a page with product information to be cached, but it should not allow a `Set-Cookie` response header (and its associated cookie) to be stored with it. On the other hand, the shopping site should suppress the caching of any pages that contain personal information, such as shipping information or the contents of a shopping basket.

2.4 Why Cookies?

Cookies make it easier to build stateful web applications, but they are not essential to achieve that purpose. To accomplish much the same thing, a server can, for example, embed state information in URLs, use hidden fields in HTML forms, or use the client's Internet Protocol (IP) address. But these approaches are failure-prone. As Section 4 describes, IP addresses are an unreliable way to identify a user or computer. If URLs or forms are used, the state information is not part of the protocol; rather it is contained within the user-accessible information that the server returns to the user. If a user clicks on a Back button in the browser, the user's state would roll back to what it had been for the earlier page. For a

[3] Suggestive domain names (such as `shop.com`) in the examples merely convey the role the names play in the example. They do not imply that anything described in such examples applies to any site that might exist with that domain name.

shopping application, this behavior would have the effect of removing items from the shopping basket. Moreover, both approaches lend themselves to mischief: a user can easily capture the text of the URL or form fields, edit it, and resubmit the information to the server, with unpredictable results. Finally, embedding state information in URLs is very unfriendly to caches, and web caches are considered valuable for reducing network traffic and, thereby, congestion.

2.5 How Are Cookies Used?

Cookies have a variety of uses, some of which are controversial. I've already described how they can make it possible (actually, strictly speaking, they make it *easier*) to implement shopping applications.

Cookies can also be used to store "login" information for sites that provide personalized access, so you don't have to keep entering your name and password each time you visit.[4]

A web site can also use cookies to track which pages you visit on the site. The site's administrators may want to use the cookies to better understand how users navigate the site. With such an understanding, they can organize the site so the most popular information is in places that are easier for users to find.

Ordinarily a web site (`example.com`) cannot distinguish a particular user over time; at best it can tell what IP address your computer has. However, that IP address often does not identify you uniquely:

—If you use an HTTP proxy, the web site will "see" the proxy's IP address, not your computer's. Thus all of the proxy's users will appear to be one user to a server.

—If you use an ISP that provides you with a temporary IP address each time you connect to it, your IP address could be different when you visit `example.com` at different times, and you would appear to the server as different users.

A cookie that's stored on your host computer is indifferent to the path by which your computer connects to `example.com`. The "cookie contract" stipulates that your computer return its cookie to `example.com` when you visit it again, regardless of what your IP address is (or which ISP you use). On a single-user computer like a PC, the cookie thus identifies the collection of all users of the computer. On a multi-user computer, the cookie identifies the user(s) of a particular account.

Identifies does not necessarily mean that `example.com` somehow knows your name, address, or other personal information. Unless you explicitly provide personal information, all that `example.com` can do is assemble a list of URLs on its site that you (or, rather, the user of a particular computer or account) have visited, as identified by a cookie. Of course, if you *do* supply personal information to `example.com`, perhaps to register for some service or to order merchandise, that information can be associated with the URLs you visited. As I'll discuss later, this ability to monitor your browsing habits and possibly to associate what you've looked at with who you are is at the heart of the privacy concerns that cookies raise.

---

[4] [Moore and Freed 2000] deprecates using cookies to store such information under some circumstances.

3. THE IETF STANDARDS PROCESS

Because the cookie specification and the HTTP specification have emerged from the Internet Engineering Task Force (IETF) standards process, it's essential to understand the IETF's hierarchical organization and how its standards process works.

The IETF evolved out of the earliest days of the Arpanet to become the *de facto* Internet standards body. At the lowest level are "members." Unlike most other standards bodies, however, where the body is an internationally sanctioned group and participants are national representatives, IETF is an open organization whose members comprise literally anyone who is willing to participate constructively in IETF activities. There are no membership cards or dues. While they are often employed by corporations with a keen interest in the ultimate shape that IETF standards take, members are expected to develop standards on their *technical merits* alone, and by custom they are assumed to speak for themselves, and not their employers.[5]

Members typically participate in one or more *working groups* of interest, such as the HTTP Working Group. A working group (WG) organizes itself, chooses one or more chairpeople, and draws up a charter, which identifies work items and a schedule by which they will be completed. Working groups are expected to have a limited lifetime, on the order of 2-3 years, although the work items in the charter usually span 12-18 months. Most of the work of a working group gets done on email mailing lists, which are open to anyone to join, and which must have a public archive. Occasionally small numbers of working group members meet face-to-face to work out particularly knotty issues. Otherwise, the only time most members actually *see* one another, if ever, is at IETF meetings, held three times a year. One or more attendees take meeting notes, which then get published on the IETF's web site after the meeting. The availability of meeting notes, open mailing lists, and list archives helps keep the process transparent to anyone interested.

The standards that the IETF produces begin life as an *Internet Draft* (I-D). Anyone may submit an I-D to the Internet Drafts Administrator of the IETF. Typical I-Ds are part of a working group's business, and some of those are *standards track*, as opposed to, say, *informational*. I-Ds usually have a six-month expiration: if no action is taken on an I-D, it vanishes. Two typical actions are for the author to submit a revised I-D to supersede the first, or for the IESG (see below) to approve an I-D to advance along the standards track.

Attendees at working group sessions at the meeting are expected to have read the applicable current Internet Drafts. The IETF sets a cut-off date for submitting I-Ds in advance of each meeting. Therefore, a flood of new I-Ds typically gets announced by the IETF just before a meeting, as authors try to make their latest version available. The cut-off ensures an adequate amount of time to review the I-Ds.

Groups of Working groups comprise *Areas*. For example, the HTTP Working Group was part of the Applications Area. Each Area has one or two *Area Directors*, who are experienced IETF members. The Area Directors as a group comprise

[5] That's not to say that there aren't differing opinions about "technical merits," which may reflect corporate interests.

the *Internet Engineering Steering Group* (IESG). IESG members administer IETF process, and they monitor the activities of the working groups and, among other things, watch for similar work in different areas that perhaps should be coordinated.

The *Internet Architecture Board* (IAB) comprises senior members of the IETF who guide the overall evolution of Internet standards and adjudicate disputes about IESG actions.

A typical IETF standard's life-cycle begins with an initial I-D. The I-D undergoes vigorous scrutiny and discussion by a working group on its mailing list, which results in a cycle of revised I-Ds and further discussion. When the working group reaches "rough consensus," the chair issues a *Last Call* for working group comments. Assuming all of those are addressed adequately by the author(s), the chair recommends that the IESG consider the I-D to be a *Proposed Standard*. The IESG then issues its own, IETF-wide, Last Call for comments. If there are comments, the author(s) will revise the I-D and restart the discussion cycle, although at this point there usually are but few comments. Once the IESG approves the I-D to be a Proposed Standard, it gets submitted to the *RFC Editor*, who edits and formats the document and assigns it an RFC (Request for Comments) number. Once published as an RFC, a document never changes. It can only be superseded. Indeed, as a specification progresses through the IETF process, a newer RFC often supersedes a previous one.

The IETF emphasizes "rough consensus and running code." Note that "consensus" does not mean "unanimity." The process requires that all voices must be heard, but not all voices must be heeded.

Interested parties are expected to implement a specification once it becomes a Proposed Standard.[6] Before the RFC can progress to the next stage of the process, *Draft Standard*, there must be evidence that at least two independently developed interoperating implementations exist. This requirement ensures, first, that the specification is clear enough that two or more implementors, working independently, interpret the specification the same way. Second, the requirement demonstrates that the pieces so-developed can actually communicate with each other, which, of course, is essential for any useful networking protocol! Finally, after further review and implementation experience, a mature specification advances to *Full Standard* (or just *Standard*).

4. RFCS 2109 AND 2965: A BRIEF HISTORY

4.1 In the Beginning...

When the World Wide Web first made its way into the public's consciousness in 1993, the web browser of choice was *Mosaic*, from The University of Illinois's National Center for Supercomputing Applications (NCSA). Mosaic offered no support for a state mechanism. Early applications that wanted or needed to support state had to find an unsatisfactory workaround, possibly among those mentioned earlier.

The first publicly available version of the Netscape Navigator browser (September, 1994) supported state management [Montulli 2001], although that fact was not well known at the time. The mechanism had been introduced at the request of one

[6] Indeed, they often begin to do so before then, and their implementation experience often provides feedback for the evolution of the specification even before it becomes an RFC.

of Netscape's customers to provide the kind of stateful mechanism we now recognize. Lou Montulli at Netscape wrote the original specification, and he chose the term "cookie." Cookies solved the problems identified earlier: applications worked correctly even in the face of a user's page navigation; and cookies were part of the protocol, not part of the content, and thus they were less accessible to a user. Applications that used cookies were more robust than those using alternatives.

It now seems hard to imagine the Internet "landscape" in April, 1995, when the cookie story begins. Corporate and government web sites had started to proliferate. The technical community actively discussed the possibilities of what we now call "e-commerce." ISPs had started to appear and to offer software bundles that incorporated Mosaic or Navigator. Some early adopters had Internet access at home.[7] More people had Internet access at work, but even email access was relatively uncommon outside the technical community. Amazon.com would not open for business for three months. No one had used the phrase "dot-com."

The current cookie standard reflects the interplay of technical issues, personalities, IETF procedures, corporate influences, and external political influences. Table 1 summarizes the important events in the standardization timeline. Although the process took a long time, note that significant "dead time" occurred following Last Calls. The Appendix recounts in considerable detail the evolution of the cookie specification through two RFC cycles. In the sections immediately below I give a condensed version that highlights the key issues.

September, 1994	Netscape Navigator 0.9 beta, includes cookie support
April, 1995	discussions begin on [www-talk]
August, 1995	`State-Info` I-D
December, 1995	state management sub-group of HTTP WG forms
February, 1996	`state-mgmt-00`, first public cookie I-D
June, 1996	working group Last Call on `state-mgmt-02`
August, 1996	IESG Last Call on `state-mgmt-03`
February, 1997	RFC 2109, *HTTP State Management Mechanism*
March, 1998	working group Last Call on `state-man-mec-08`
June, 1999	IESG Last Call on `state-man-mec-10`
April, 2000	new IESG Last Call on `state-man-mec-12`
October, 2000	RFC 2965, *HTTP State Management Mechanism*
	RFC 2964, *Use of HTTP State Management*

Table 1. Timeline of HTTP State Management Standardization

4.2 RFC 2109: December, 1995 to February, 1997

By late 1995, three proposals for adding state to HTTP were circulating in the technical community. Because the HTTP Working Group was more concerned with producing an HTTP/1.1 specification to solve urgent needs, Larry Masinter, as chair of the group, asked the parties interested in state management to form a sub-group to recommend a single approach to the rest of the WG. As author of one such proposal, I agreed to head up the "state sub-group," and a group of eight

[7] America Online (AOL), with two million subscribers, had yet to offer direct Internet access, and did not until October, 1997. AOL now has over 29 million subscribers.

people, including Lou Montulli, the author of Netscape's specification [NS], began to meet by email and conference call. After considering the alternatives, we soon decided to adopt Netscape's underlying mechanism, while preparing a more precise specification.

[NS] provides rules whereby a cookie can be shared among multiple servers, based on their domain names. We identified two problems that this cookie sharing mechanism could enable[8]: 1) Cookies can "leak" to servers other than those intended by the originating server. 2) A server in a domain can cause a denial-of-service attack, either inadvertently or intentionally, by sending cookies that will disrupt an application that runs on another server in the same domain ("cookie spoofing"). We worded the specification to try to minimize how widely cookies could proliferate, subject to the (implicit) constraint that the control be based on domain names.

In February, 1996, we identified what we felt was a considerable threat to privacy, *third-party cookies*, or "unverifiable transactions." [9] A transaction, or request, is "verifiable" when the user can tell beforehand where it will go. A browser can receive third-party cookies if it loads a page from one web site, loads images (such as ads) from another web site, and the latter web site sends a cookie with the image. Our concern was that, whereas a user could well expect a cookie from the first web site, she has no reason to expect, or even to know, that her browser will visit another web site (through an unverifiable transaction) and receive a cookie from it. We added wording to the specification that either outright prohibited a browser from accepting third-party cookies ("cookies in unverifiable transactions"), or that permitted a browser to accept them, provided they were controlled by a user-controlled option whose default value was to reject them.

By late April, 1996, the sub-group had prepared an I-D for review by the entire WG. After some revisions, there was a WG Last Call in June, some small revisions, and an IESG Last Call in early August. In October, the IESG expressed concern that, in essence, suggested the specification was too lenient with respect to identifying which transactions were "verifiable." Keith Moore, an Applications Area Director, and I worked out compromise wording with the IESG to convey the idea that the inspection mechanism described in the specification was at best minimally acceptable. In December, with this change, plus another, minor one, the IESG approved the specification to be published as an RFC, which it was in February, 1997, as RFC 2109, [Kristol and Montulli 1997] "HTTP State Management."

Some of the threads common to the evolution of the specification had already manifested themselves:

—We were concerned about how cookies' domain names affected their ability to proliferate beyond their intended (or desired) targets.

—We had noted a potential privacy threat in "third-party cookies."

—The IESG pushed for even stricter language regarding "third-party cookies" than the WG felt was feasible, given constraints of compatibility and what could reasonably be demanded of an implementation.

—At a particularly volatile time in the evolution of Web technology, IETF process

[8]See Section A.4.1 for more details.
[9]More detail can be found in Section A.4.4.

roughly doubled the time between the specification's being accepted by the WG and the time it appeared as an RFC.

4.3 RFC 2965: February, 1997 to October, 2000

RFC 2109 attempted to extend [NS] while using the same HTTP headers. The hope was that already-deployed clients and servers could be upgraded incrementally to use the new specification. However, around the time that the IESG approved RFC 2109, but before it got published, a compatibility issue surfaced. We found that Netscape Navigator and Microsoft Internet Explorer (MSIE) behaved differently in the face of the attributes we had introduced as part of RFC 2109. Clearly the WG would have to revise the specification.

4.3.1 *Fixing the Incompatibility.* Because the two extant major browsers disagreed on how to treat unfamiliar attributes, we were inexorably led to introduce one or more new headers to resolve the problem. We discussed several different approaches, all of which entailed putting the "new" attributes in a new header, where they would not confuse the code that handles existing headers.[10]

4.3.2 *Unverifiable Transactions and Certified Cookies.* The publication of RFC 2109 resulted in articles about cookies appearing in the trade and popular press and to heated protests from the Web advertising networks that emerged while the RFC was being written and discussed. Because many of the networks had developed business models that relied on third-party cookies to do targeted advertising, they felt the RFC's mandate to disable third-party cookies by default was a threat to their business. However, the WG generally supported the RFC's default, noting that the RFC's restrictions on third-party cookies would affect the advertisers' *business models* that relied on tracking users, not the advertising business itself.

These discussions about third-party cookies led to a proposal of "certified cookies." A certified cookie would assert how the web server would use the cookie, and it would be signed cryptographically by an auditing agency. A user could configure her browser to specify what kinds of uses of cookies she is comfortable with, and the browser would automatically accept or reject cookies, whether they were from third parties or not, based on the configuration. The WG's goal was to layer the certified cookies mechanism on top of the regular cookie mechanism to enhance the (default) third-party cookie ban and other cookie controls.

4.3.3 *Deadlock and Resolution.* Through early 1997, the WG attempted to resolve the issues described above (and others; see Section B). By August, however, discussions had become circular, repeating earlier arguments and mingling technical and social (privacy) issues. We were making no progress.

As a way out of the impasse, we embarked on a two-part strategy to proceed. I would remove, temporarily, the parts of the specification concerning "unverifiable transactions," letting us focus on the purely technical part. Once we agreed on the technical part, we would re-introduce the political part and try to reach further consensus, at which point we should be done.

By February, 1998, we had achieved consensus on the technical part of the speci-

[10] More complete discussion is in Section B.1.

fication. When I subsequently added back the "unverifiable transactions" language, surprisingly there were no further comments, and Working Group and IESG Last Calls quickly followed. However the resulting specification, with minor modifications, languished for two years. Apparently the IESG felt the need to set stronger guidelines for the use of cookies than the prospective (new) RFC contained.[11]. Only when this set of guidelines ([Moore and Freed 2000]) was written and accepted could the cookie specification be published as RFC 2965.

5. PRIVACY/POLITICS

The cookie specification may have been the first IETF standard at the intersection of technology and privacy to get widespread public notice, some of which I'll describe below. As the Internet moved from research plaything to public plaything to vital communications infrastructure, the IESG began to expect all RFCs to include a thoughtful "Security Considerations" section.[12] Privacy was considered an element of security in this context. Indeed, the longest delays incurred during the standardization process of the two cookie RFCs were due to the tension between the IESG, which pushed for *even stricter* privacy safeguards than those two RFCs contained, and the HTTP Working Group, which could achieve even rough consensus only with slightly weaker safeguards.

5.1 Federal Trade Commission

The U.S. Federal Trade Commission (FTC) convened a workshop on consumer privacy in June, 1996. Among the topics discussed was the possible use of the World Wide Web Consortium's (W3C) Platform for Internet Content Selection (PICS) [PICS2000] "to facilitate automatic disclosure of privacy policies and the availability of consumer choice regarding the use of personal information."[FTC 1996]

In March, 1997, just weeks after RFC 2109 appeared, the FTC announced another Consumer Information Privacy Workshop to be held in June [FTC 1997]. Among the topics discussed were consumer online privacy, industry self-regulation, and technology that could be used to enhance privacy. In a comment letter to the FTC regarding the workshop, Peter F. Hartley, Netscape's Global Public Policy Counsel wrote:

> 2.14 Interactive technology has evolved since June 1996 to address many of the privacy concerns expressed regarding cookies and what information was placed on a user's computer and with what notice and consent. Software manufacturers and open technical standards bodies have produced innovations that enable users to have more control over cookies and how web site operators are able to place information on one's computer. At this point in time many web site operators and related third parties are reviewing the technical standards concerning cookies. These improvements and changes in cookie technology will be

[11] The new RFC did not differ materially from RFC 2109 in its privacy provisions. However, the composition of the IESG had changed in the intervening 3 1/2 years.

[12] [Postel 1993], superseded by [Postel and Reynolds 1997], which came *after* RFC 2109, called for a "Security Considerations" section, but many RFCs said there were no security issues.